US009110930B2

(12) United States Patent

Archer et al.

(10) **Patent No.: US 9,110,930 B2**

(45) **Date of Patent: Aug. 18, 2015**

(54) **PARALLEL APPLICATION CHECKPOINT IMAGE COMPRESSION**

(71) Applicant: **International Business Machines Corporation**, Armonk, NY (US)

(72) Inventors: **Charles J. Archer**, Rochester, MN (US); **Benjamin E. Lynam**, Seattle, WA (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 65 days.

(21) Appl. No.: **13/973,376**

(22) Filed: **Aug. 22, 2013**

(65) **Prior Publication Data**

US 2015/0055889 A1 Feb. 26, 2015

(51) **Int. Cl.**

G06F 17/30 (2006.01)
G06K 9/62 (2006.01)
G06F 11/34 (2006.01)

(52) **U.S. Cl.**

CPC ***G06F 17/3028*** (2013.01); ***G06F 11/3404*** (2013.01); ***G06K 9/6202*** (2013.01); ***G06K 9/6282*** (2013.01)

(58) **Field of Classification Search**

CPC ... G06F 9/3822; G06F 9/3885; G06K 9/6202; G06K 9/6282

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,469,562	A *	11/1995	Saether	714/20
5,481,699	A *	1/1996	Saether	714/15
5,864,849	A *	1/1999	Bohannon et al.	707/648
6,343,313	B1 *	1/2002	Salesky et al.	709/204
6,795,966	B1 *	9/2004	Lim et al.	718/1
7,096,392	B2 *	8/2006	Sim-Tang	714/48
7,293,200	B2 *	11/2007	Neary et al.	714/35
7,363,549	B2 *	4/2008	Sim-Tang	714/48
7,478,278	B2 *	1/2009	Archer et al.	714/15
7,487,393	B2 *	2/2009	Archer et al.	714/15
7,627,783	B2 *	12/2009	Archer et al.	714/15
7,680,834	B1 *	3/2010	Sim-Tang	707/999.201

(Continued)

OTHER PUBLICATIONS

Sankaran et al., "The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing", In Proceedings, Los Alamos Computer Science Institute (LACSI) Symposium, Oct. 2003, pp. 1-12, LACSI, Rice University, Houston.

(Continued)

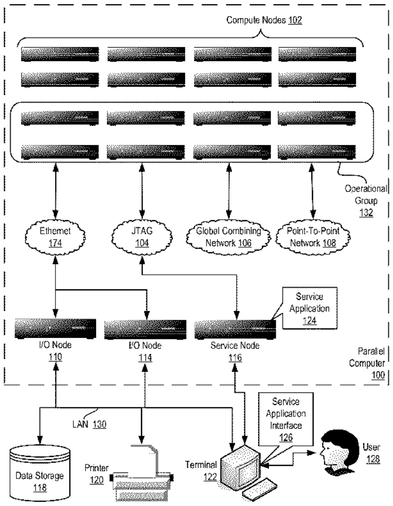
Primary Examiner — Manav Seth

(74) *Attorney, Agent, or Firm* — Edward J. Lenart; Kennedy Lenart Spraggins LLP

(57) ABSTRACT

Parallel application checkpoint image compression may be carried out in a parallel computer. The parallel computer may include a plurality of compute nodes, where each node is configured to execute one or more parallel tasks of the parallel application. The parallel tasks may be organized into an operational group for collective communications. In such a parallel computer, checkpoint image compression may include: generating, by each task of the parallel application, an image for checkpointing the parallel application; selecting, by an image management task, one of the images as a base template image; constructing, by the image management task, a binary radix tree, including storing differences between each task's image and the base template image in the binary radix tree; and storing, by the image management task as a checkpoint for the parallel application, the binary radix tree and the base template image, without storing every task's image.

18 Claims, 10 Drawing Sheets



(56) References Cited

U.S. PATENT DOCUMENTS

7,788,521	B1 *	8/2010	Sim-Tang	714/4.12
8,151,140	B2 *	4/2012	Sim-Tang	714/20
8,307,243	B2 *	11/2012	Archer et al.	714/20
8,352,482	B2	1/2013	Hansen	
2006/0236152	A1 *	10/2006	Archer et al.	714/16
2008/0178177	A1 *	7/2008	Archer et al.	718/100
2008/0313506	A1 *	12/2008	Archer et al.	714/48

OTHER PUBLICATIONS

Sankaran et al., "Parallel Checkpoint/Restart for MPI Applications", In Proceedings, Los Alamos Computer Science Institute (LACSI) Symposium, Oct. 2003, pp. 1-12, LACSI, Rice University, Houston.

Hursey et al., "The Design and Implementation of Checkpoint/Restart Process Fault Tolerance for Open MPI", Proceedings of the 21st IEEE International Parallel and Distributed Processing (IPDPS) Symposium, Mar. 2007, pp. 1-8, Institute of Electrical and Electronics Engineers, Inc. (IEEE), USA.

Nicolae et al., "BlobCR: Efficient Checkpoint-Restart for HPC Applications on IaaS Clouds using Virtual Disk Image Snapshots", Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (2011), Series SC '11, Article 34, Aug. 16, 2011, pp. 1-12, ACM, New York, NY, USA. DOI : 10.1145/2063384.2063429.

* cited by examiner

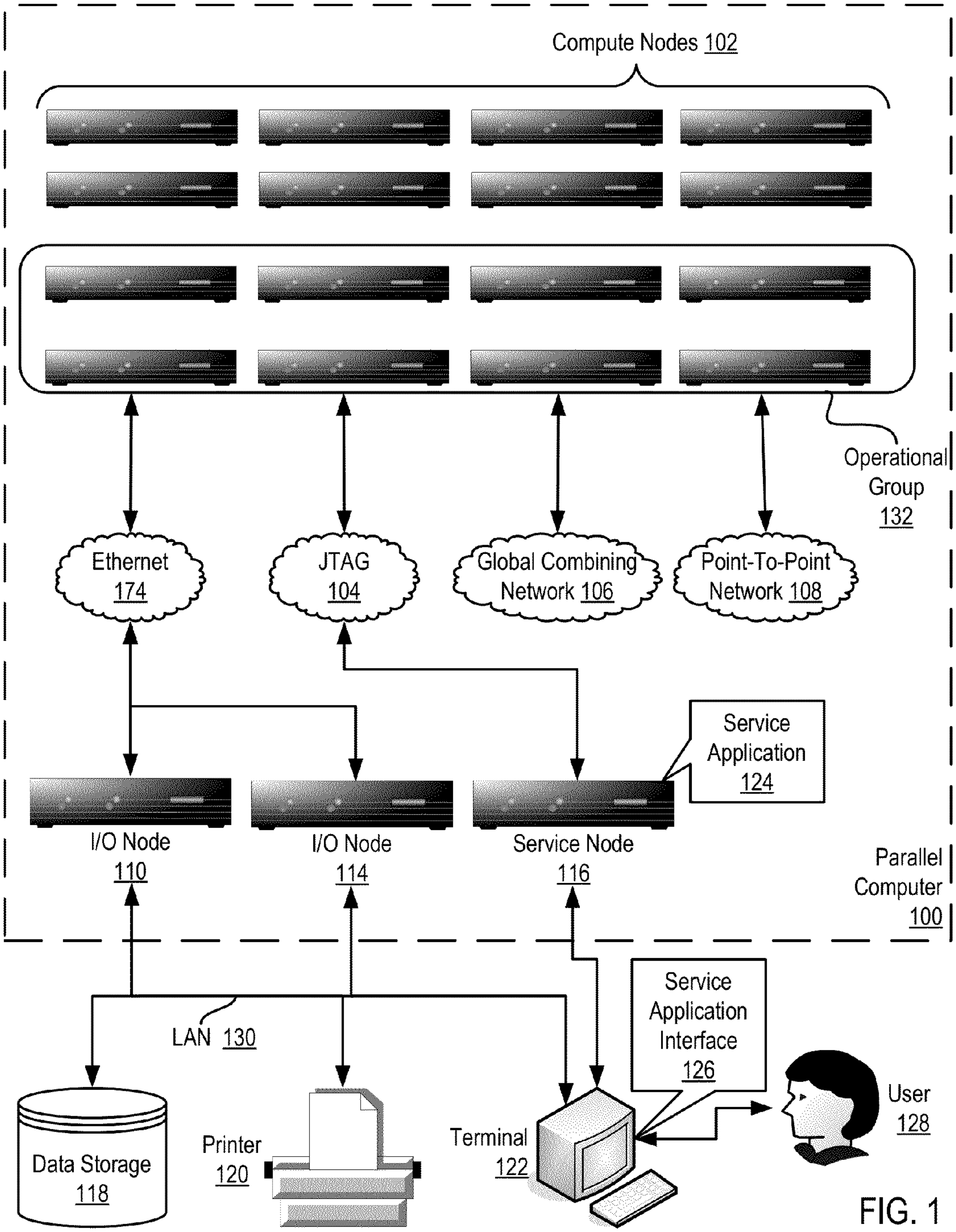

Compute Nodes 102
Operational Group 132
Ethernet 174
JTAG 104
Global Combining Network 106
Point-To-Point Network 108
Service Application 124
I/O Node 110
I/O Node 114
Service Node 116
Parallel Computer 100
Service Application Interface 126
LAN 130
User 128
Data Storage 118
Printer 120
Terminal 122

FIG. 1

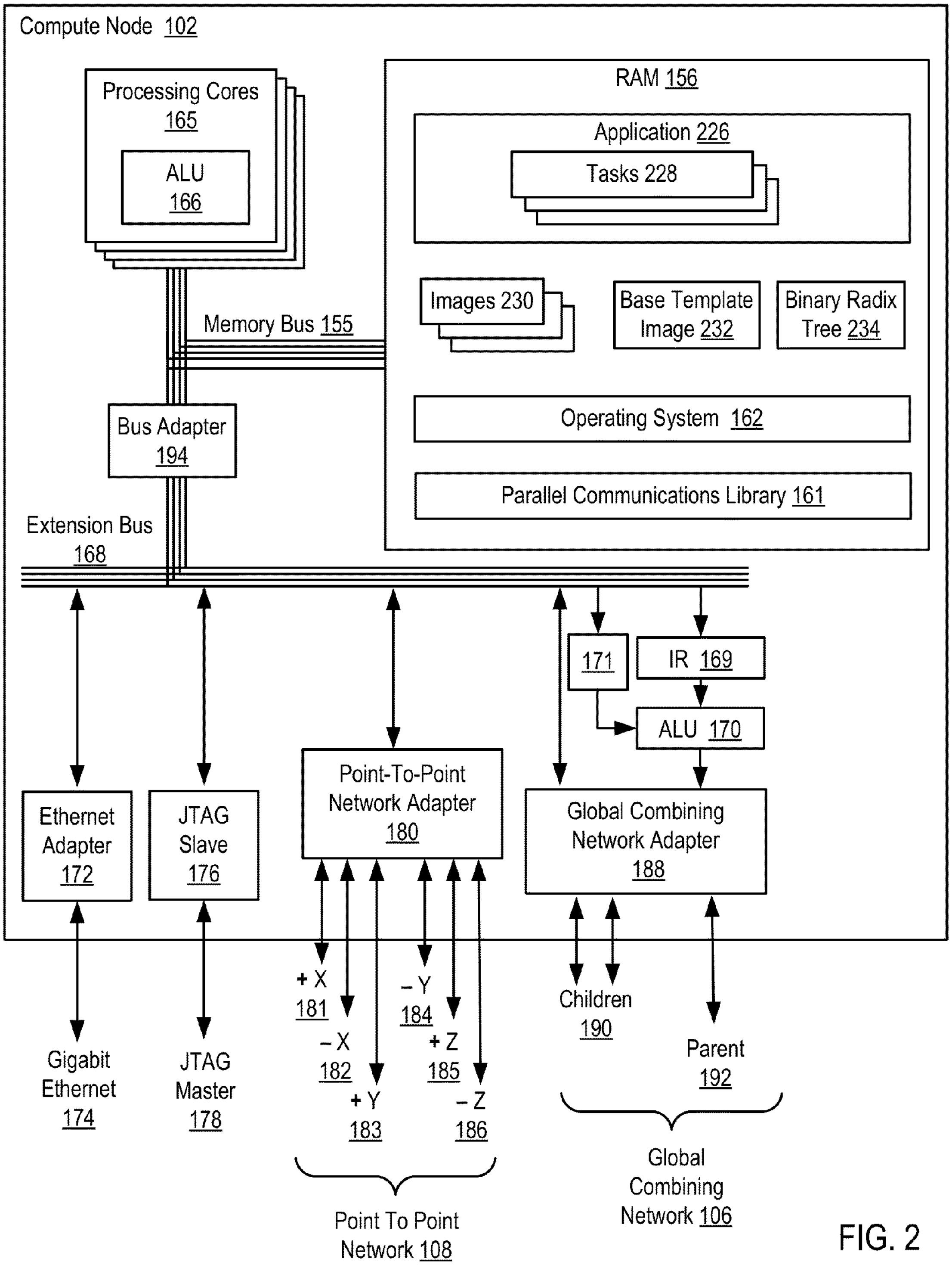

Compute Node 102
Processing Cores 165
ALU 166
RAM 156
Application 226
Tasks 228
Images 230
Base Template Image 232
Binary Radix Tree 234
Operating System 162
Parallel Communications Library 161
Memory Bus 155
Bus Adapter 194
Extension Bus 168
171
IR 169
ALU 170
Ethernet Adapter 172
JTAG Slave 176
Point-To-Point Network Adapter 180
Global Combining Network Adapter 188
+ X 181
− X 182
+ Y 183
− Y 184
+ Z 185
− Z 186
Children 190
Parent 192
Gigabit Ethernet 174
JTAG Master 178
Point To Point Network 108
Global Combining Network 106

FIG. 2

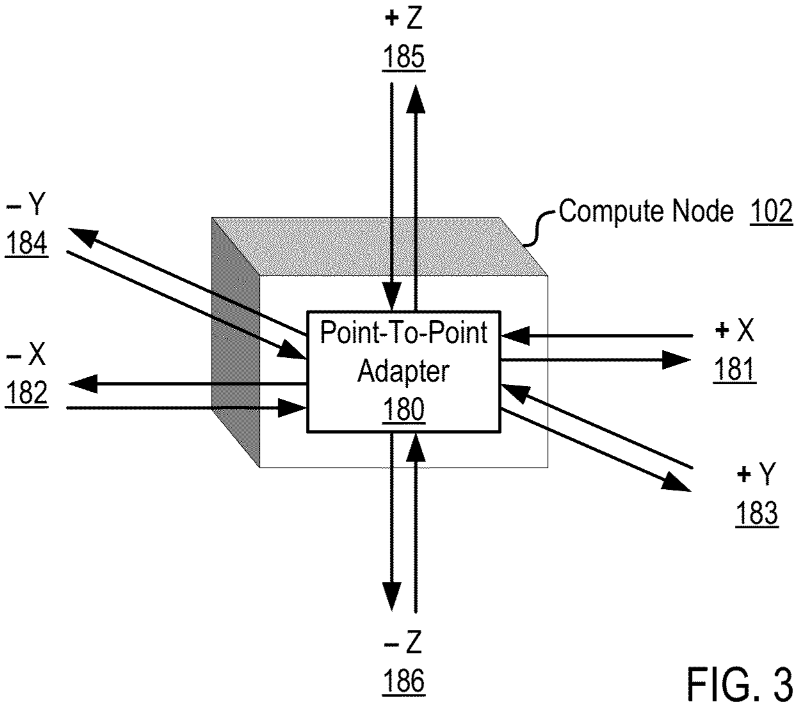

+ Z
185
– Y
184
Compute Node 102
Point-To-Point
Adapter
180
+ X
181
– X
182
+ Y
183
– Z
186

FIG. 3

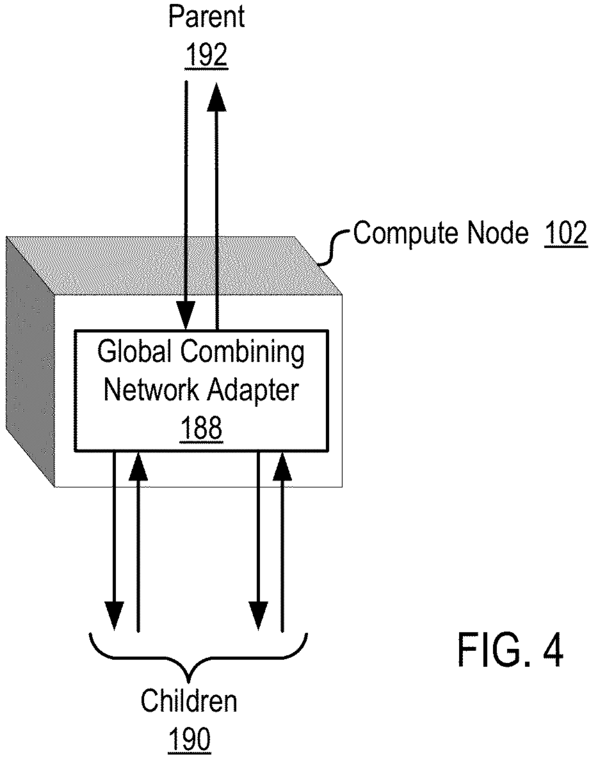

Parent
192
Compute Node 102
Global Combining Network Adapter
188
Children
190

FIG. 4

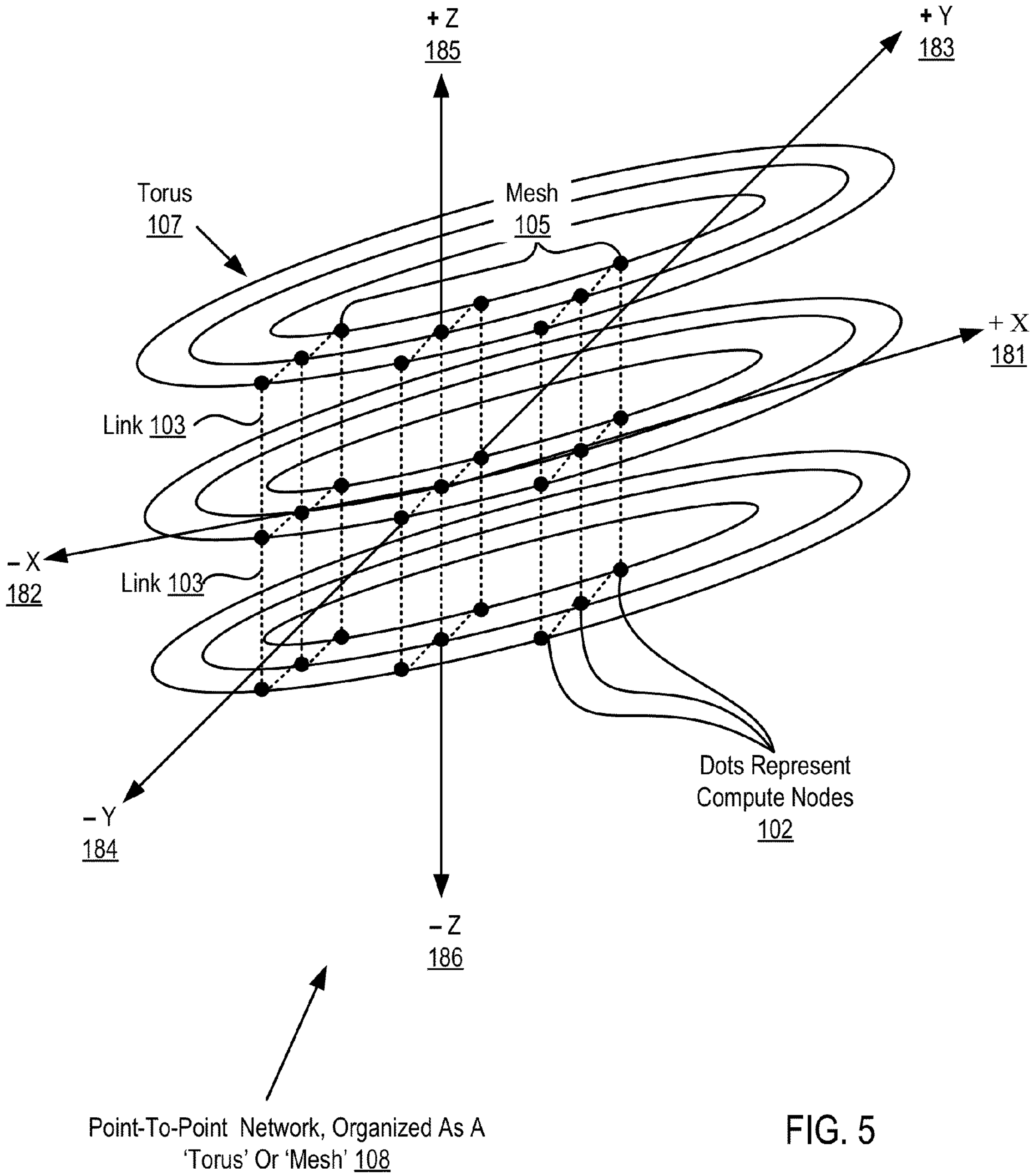

+ Z
185
+ Y
183
Torus
107
Mesh
105
+ X
181
Link 103
Link 103
– X
182
– Y
184
Dots Represent
Compute Nodes
102
– Z
186
Point-To-Point Network, Organized As A
'Torus' Or 'Mesh' 108

FIG. 5

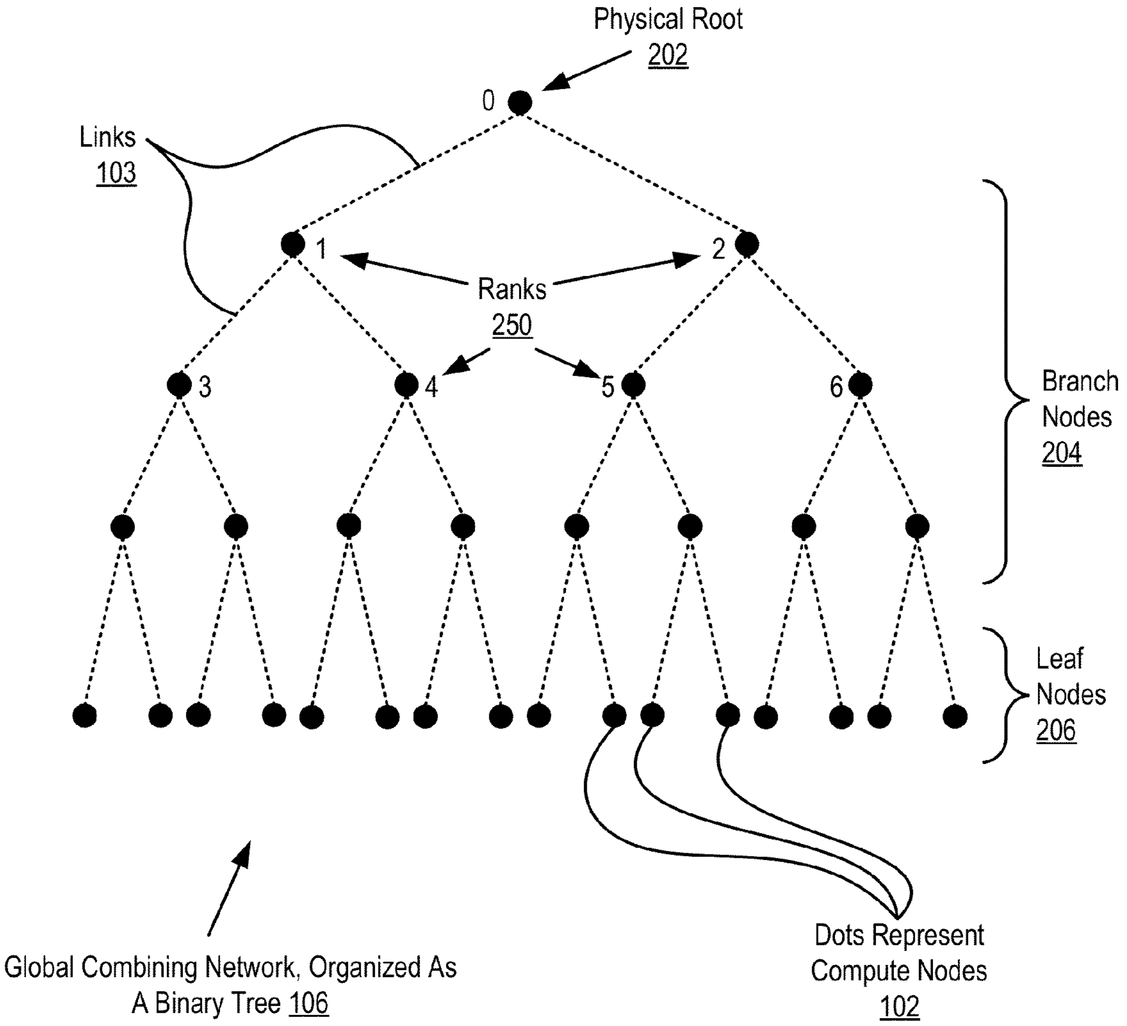

Physical Root
202
0
Links
103
1
2
Ranks
250
3
4
5
6
Branch
Nodes
204
Leaf
Nodes
206
Global Combining Network, Organized As
A Binary Tree 106
Dots Represent
Compute Nodes
102

FIG. 6

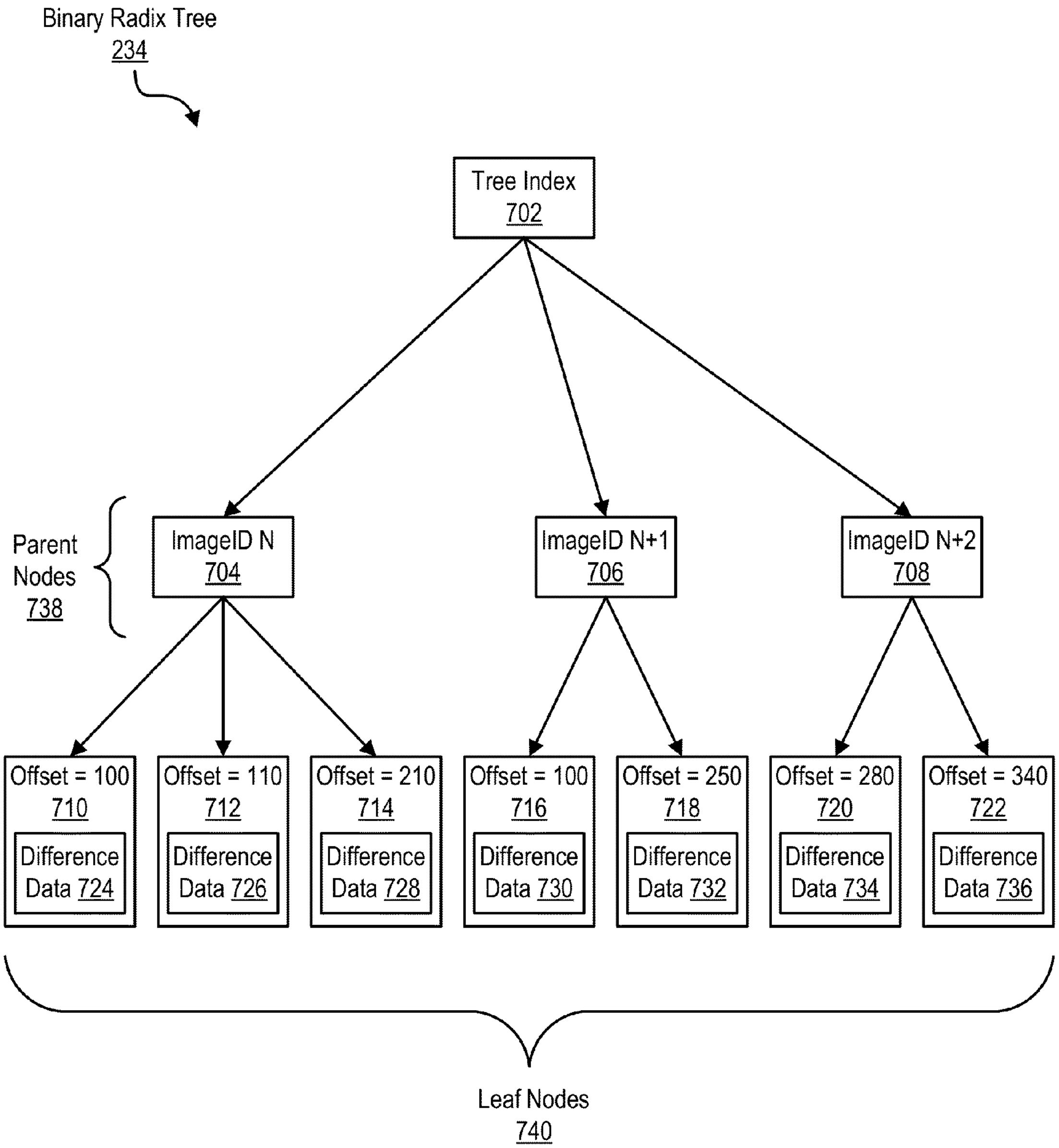

Binary Radix Tree
234
Tree Index
702
Parent
Nodes
738
ImageID N
704
ImageID N+1
706
ImageID N+2
708
Offset = 100
710
Difference
Data 724
Offset = 110
712
Difference
Data 726
Offset = 210
714
Difference
Data 728
Offset = 100
716
Difference
Data 730
Offset = 250
718
Difference
Data 732
Offset = 280
720
Difference
Data 734
Offset = 340
722
Difference
Data 736
Leaf Nodes
740

FIG. 7

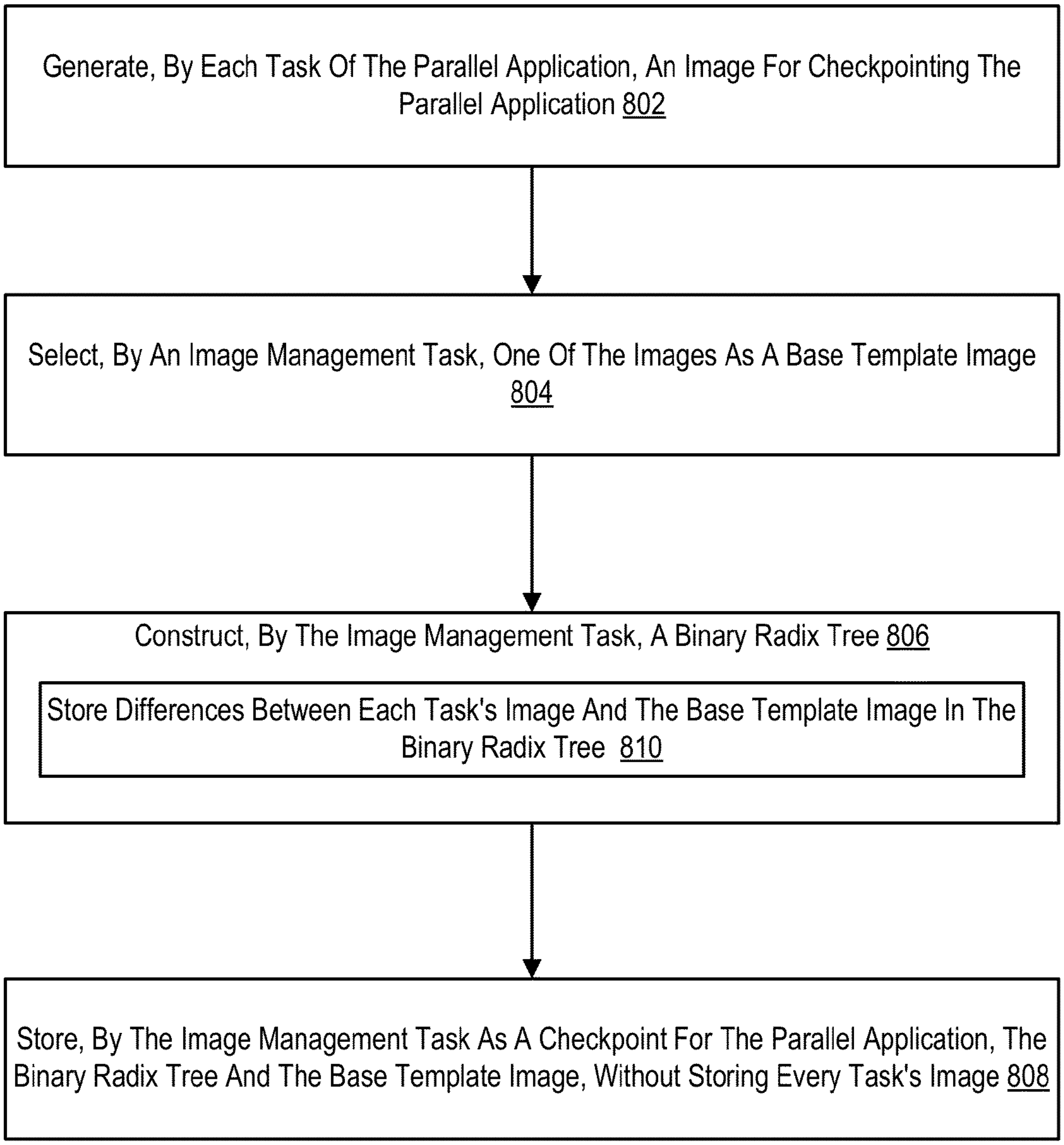

Generate, By Each Task Of The Parallel Application, An Image For Checkpointing The Parallel Application 802
Select, By An Image Management Task, One Of The Images As A Base Template Image 804
Construct, By The Image Management Task, A Binary Radix Tree 806
Store Differences Between Each Task's Image And The Base Template Image In The Binary Radix Tree 810
Store, By The Image Management Task As A Checkpoint For The Parallel Application, The Binary Radix Tree And The Base Template Image, Without Storing Every Task's Image 808

FIG. 8

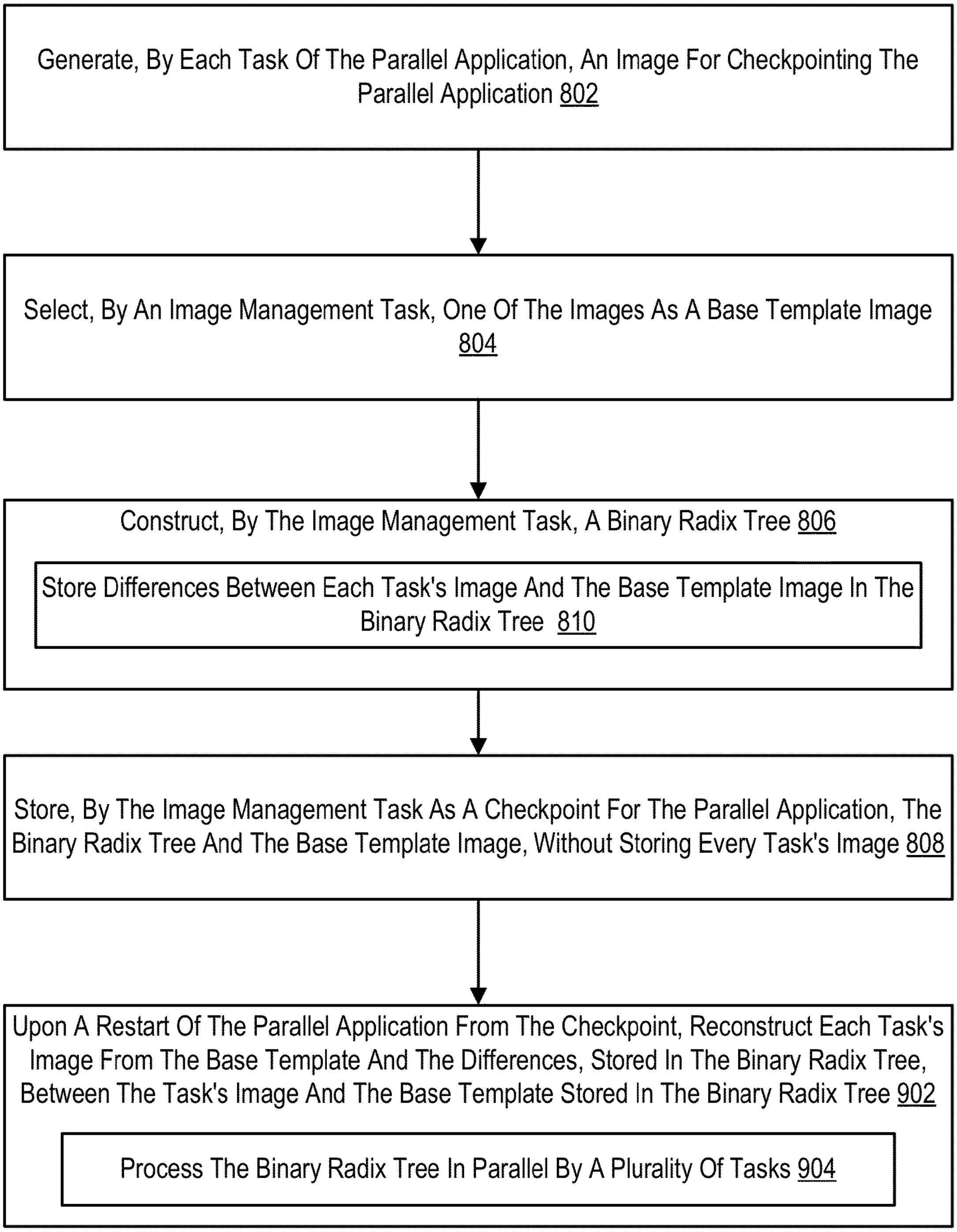

Generate, By Each Task Of The Parallel Application, An Image For Checkpointing The Parallel Application 802
Select, By An Image Management Task, One Of The Images As A Base Template Image 804
Construct, By The Image Management Task, A Binary Radix Tree 806
Store Differences Between Each Task's Image And The Base Template Image In The Binary Radix Tree 810
Store, By The Image Management Task As A Checkpoint For The Parallel Application, The Binary Radix Tree And The Base Template Image, Without Storing Every Task's Image 808
Upon A Restart Of The Parallel Application From The Checkpoint, Reconstruct Each Task's Image From The Base Template And The Differences, Stored In The Binary Radix Tree, Between The Task's Image And The Base Template Stored In The Binary Radix Tree 902
Process The Binary Radix Tree In Parallel By A Plurality Of Tasks 904

FIG. 9

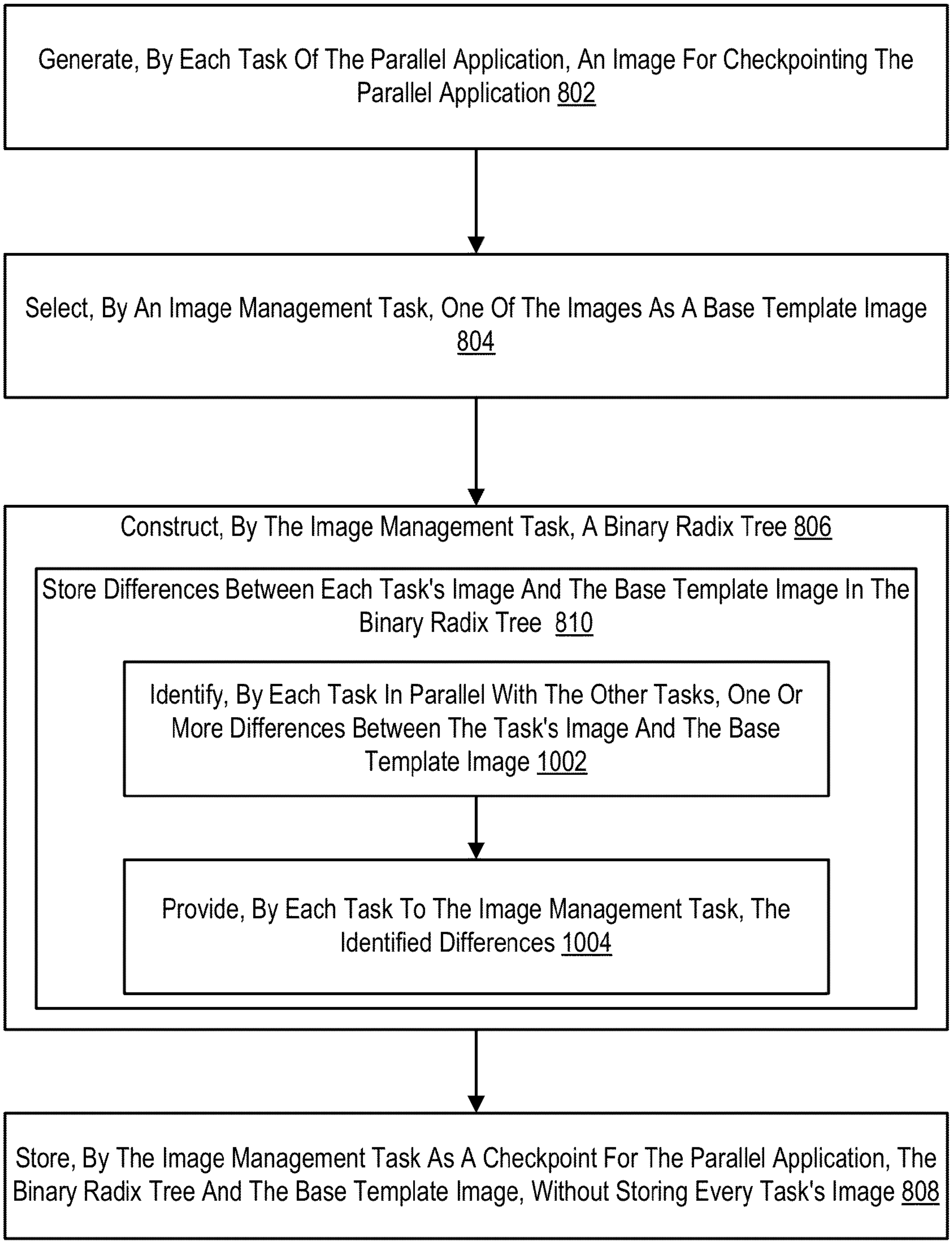

Generate, By Each Task Of The Parallel Application, An Image For Checkpointing The Parallel Application 802
Select, By An Image Management Task, One Of The Images As A Base Template Image 804
Construct, By The Image Management Task, A Binary Radix Tree 806
Store Differences Between Each Task's Image And The Base Template Image In The Binary Radix Tree 810
Identify, By Each Task In Parallel With The Other Tasks, One Or More Differences Between The Task's Image And The Base Template Image 1002
Provide, By Each Task To The Image Management Task, The Identified Differences 1004
Store, By The Image Management Task As A Checkpoint For The Parallel Application, The Binary Radix Tree And The Base Template Image, Without Storing Every Task's Image 808

FIG. 10

PARALLEL APPLICATION CHECKPOINT IMAGE COMPRESSION

BACKGROUND OF THE INVENTION

1. Field of the Invention

The field of the invention is data processing, or, more specifically, methods, apparatus, and products for parallel application checkpoint image compression.

2. Description of Related Art

The development of the EDVAC computer system of 1948 is often cited as the beginning of the computer era. Since that time, computer systems have evolved into extremely complicated devices. Today's computers are much more sophisticated than early systems such as the EDVAC. Computer systems typically include a combination of hardware and software components, application programs, operating systems, processors, buses, memory, input/output devices, and so on. As advances in semiconductor processing and computer architecture push the performance of the computer higher and higher, more sophisticated computer software has evolved to take advantage of the higher performance of the hardware, resulting in computer systems today that are much more powerful than just a few years ago.

Parallel computing is an area of computer technology that has experienced advances. Parallel computing is the simultaneous execution of the same task (split up and specially adapted) on multiple processors in order to obtain results faster. Parallel computing is based on the fact that the process of solving a problem usually can be divided into smaller tasks, which may be carried out simultaneously with some coordination.

Parallel computers execute parallel algorithms. A parallel algorithm can be split up to be executed a piece at a time on many different processing devices, and then put back together again at the end to get a data processing result. Some algorithms are easy to divide up into pieces. Splitting up the job of checking all of the numbers from one to a hundred thousand to see which are primes could be done, for example, by assigning a subset of the numbers to each available processor, and then putting the list of positive results back together. In this specification, the multiple processing devices that execute the individual pieces of a parallel program are referred to as 'compute nodes.' A parallel computer is composed of compute nodes and other processing nodes as well, including, for example, input/output ('I/O') nodes, and service nodes.

Parallel algorithms are valuable because it is faster to perform some kinds of large computing tasks via a parallel algorithm than it is via a serial (non-parallel) algorithm, because of the way modern processors work. It is far more difficult to construct a computer with a single fast processor than one with many slow processors with the same throughput. There are also certain theoretical limits to the potential speed of serial processors. On the other hand, every parallel algorithm has a serial part and so parallel algorithms have a saturation point. After that point adding more processors does not yield any more throughput but only increases the overhead and cost.

Parallel algorithms are designed also to optimize one more resource the data communications requirements among the nodes of a parallel computer. There are two ways parallel processors communicate, shared memory or message passing. Shared memory processing needs additional locking for the data and imposes the overhead of additional processor and bus cycles and also serializes some portion of the algorithm.

Message passing processing uses high-speed data communications networks and message buffers, but this communication adds transfer overhead on the data communications networks as well as additional memory need for message buffers and latency in the data communications among nodes. Designs of parallel computers use specially designed data communications links so that the communication overhead will be small but it is the parallel algorithm that decides the volume of the traffic.

Many data communications network architectures are used for message passing among nodes in parallel computers. Compute nodes may be organized in a network as a 'torus' or 'mesh,' for example. Also, compute nodes may be organized in a network as a tree. A torus network connects the nodes in a three-dimensional mesh with wrap around links. Every node is connected to its six neighbors through this torus network, and each node is addressed by its x,y,z coordinate in the mesh. In such a manner, a torus network lends itself to point to point operations. In a tree network, the nodes typically are connected into a binary tree: each node has a parent, and two children (although some nodes may only have zero children or one child, depending on the hardware configuration). Although a tree network typically is inefficient in point to point communication, a tree network does provide high bandwidth and low latency for certain collective operations, message passing operations where all compute nodes participate simultaneously, such as, for example, an allgather operation. In computers that use a torus and a tree network, the two networks typically are implemented independently of one another, with separate routing circuits, separate physical links, and separate message buffers.

In some parallel computers, each compute node may execute one or more tasks—a process of execution for a parallel application. Each tasks may include a number of endpoints. Each endpoint is a data communications endpoint that supports communications among many other endpoints and tasks. In this way, endpoints support collective operations in a parallel computer by supporting the underlying message passing responsibilities carried out during a collective operation. In some parallel computers, each compute node may execute a single tasks including a single endpoint. For example, a parallel computer that operates with the Message Passing Interface ('MPI') described below in more detail may execute a single rank on each compute node of the parallel computer. In such implementations, the terms task, endpoint, and rank are effectively synonymous.

In some parallel computer each task of a parallel application may be configured to create a checkpoint image from which to restart at a later time. So called 'checkpointing' creates an image of the task at the time of the checkpoint is initiated. Other terms, such as 'snapshot,' may also refer to a checkpoint. At times, the terms checkpoint, image, checkpoint image, snapshot, and the like are used synonymously. In instances in which a parallel application includes hundreds or thousands of tasks, the number of images created may be quite large and storage for such images may be limited. Further, such images, being created by separate instances of a the same tasks, may have very many similarities. As such, storing many separate images, each of which includes an amount of data that is identical to all of the other images inefficiently utilizes storage.

SUMMARY OF THE INVENTION

Methods, apparatus, and products for parallel application checkpoint image compression in a parallel computer are disclosed in this specification. The parallel computer includes a plurality of compute nodes, where each of the compute nodes is configured to execute one or more parallel tasks of the parallel application. The parallel tasks may be organized into an operational group for collective operations. Compressing a checkpoint image for the parallel application in accordance with embodiments of the present invention includes: generating, by each task of the parallel application, an image for checkpointing the parallel application; selecting, by an image management task, one of the images as a base template image; constructing, by the image management task, a binary radix tree, including storing differences between each task's image and the base template image in the binary radix tree; and storing, by the image management task as a checkpoint for the parallel application, the binary radix tree and the base template image, without storing every task's image.

The foregoing and other objects, features and advantages of the invention will be apparent from the following more particular descriptions of exemplary embodiments of the invention as illustrated in the accompanying drawings wherein like reference numbers generally represent like parts of exemplary embodiments of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates an exemplary system for parallel application checkpoint image compression according to embodiments of the present invention.

FIG. **2** sets forth a block diagram of an example compute node useful in a parallel computer capable of parallel application checkpoint image compression according to embodiments of the present invention.

FIG. **3** sets forth a block diagram of an example Point-To-Point Adapter useful in systems for parallel application checkpoint image compression according to embodiments of the present invention.

FIG. **4** sets forth a block diagram of an example Global Combining Network Adapter useful in systems for parallel application checkpoint image compression according to embodiments of the present invention.

FIG. **5** sets forth a line drawing illustrating an example data communications network optimized for point-to-point operations useful in systems capable of parallel application checkpoint image compression according to embodiments of the present invention.

FIG. **6** sets forth a line drawing illustrating an example global combining network useful in systems capable of parallel application checkpoint image compression according to embodiments of the present invention.

FIG. **7** sets forth a line drawing illustrating an example binary radix tree useful in parallel application checkpoint image compression according to embodiments of the present invention.

FIG. **8** sets forth a flow chart illustrating an example method for parallel application checkpoint image compression according to embodiments of the present invention.

FIG. **9** sets forth a flow chart illustrating another example method for parallel application checkpoint image compression according to embodiments of the present invention.

FIG. **10** sets forth a flow chart illustrating another example method for parallel application checkpoint image compression according to embodiments of the present invention.

DETAILED DESCRIPTION OF EXEMPLARY EMBODIMENTS

Exemplary methods, apparatus, and products for parallel application checkpoint image compression in a parallel computer in accordance with the present invention are described with reference to the accompanying drawings, beginning with FIG. **1**. FIG. **1** illustrates an exemplary system for parallel application checkpoint image compression according to embodiments of the present invention. The system of FIG. **1** includes a parallel computer (**100**), non-volatile memory for the computer in the form of a data storage device (**118**), an output device for the computer in the form of a printer (**120**), and an input/output device for the computer in the form of a computer terminal (**122**).

The parallel computer (**100**) in the example of FIG. **1** includes a plurality of compute nodes (**102**). The compute nodes (**102**) are coupled for data communications by several independent data communications networks including a high speed Ethernet network (**174**), a Joint Test Action Group ('JTAG') network (**104**), a global combining network (**106**) which is optimized for collective operations using a binary tree network topology, and a point-to-point network (**108**), which is optimized for point-to-point operations using a torus network topology. The global combining network (**106**) is a data communications network that includes data communications links connected to the compute nodes (**102**) so as to organize the compute nodes (**102**) as a binary tree. Each data communications network is implemented with data communications links among the compute nodes (**102**). The data communications links provide data communications for parallel operations among the compute nodes (**102**) of the parallel computer (**100**).

The compute nodes (**102**) of the parallel computer (**100**) are organized into at least one operational group (**132**) of compute nodes for collective parallel operations on the parallel computer (**100**). Each operational group (**132**) of compute nodes is the set of compute nodes upon which a collective parallel operation executes. Each compute node in the operational group (**132**) is assigned a unique rank that identifies the particular compute node in the operational group (**132**). Collective operations are implemented with data communications among the compute nodes of an operational group. Collective operations are those functions that involve all the compute nodes of an operational group (**132**). A collective operation is an operation, a message-passing computer program instruction that is executed simultaneously, that is, at approximately the same time, by all the compute nodes in an operational group (**132**) of compute nodes. Such an operational group (**132**) may include all the compute nodes (**102**) in a parallel computer (**100**) or a subset all the compute nodes (**102**). Collective operations are often built around point-to-point operations. A collective operation requires that all processes on all compute nodes within an operational group (**132**) call the same collective operation with matching arguments. A 'broadcast' is an example of a collective operation for moving data among compute nodes of an operational group. A 'reduce' operation is an example of a collective operation that executes arithmetic or logical functions on data distributed among the compute nodes of an operational group (**132**). An operational group (**132**) may be implemented as, for example, an MPI 'communicator.'

'MPI' refers to 'Message Passing Interface,' a prior art parallel communications library, a module of computer program instructions for data communications on parallel computers. Examples of prior-art parallel communications libraries that may be improved for use in systems configured according to embodiments of the present invention include MPI and the 'Parallel Virtual Machine' ('PVM') library. PVM was developed by the University of Tennessee, The Oak Ridge National Laboratory and Emory University. MPI is promulgated by the MPI Forum, an open group with representatives from many organizations that define and maintain the MPI standard. MPI at the time of this writing is a de facto standard for communication among compute nodes running a parallel program on a distributed memory parallel computer. This specification sometimes uses MPI terminology for ease of explanation, although the use of MPI as such is not a requirement or limitation of the present invention.

Some collective operations have a single originating or receiving process running on a particular compute node in an operational group (**132**). For example, in a 'broadcast' collective operation, the process on the compute node that distributes the data to all the other compute nodes is an originating process. In a 'gather' operation, for example, the process on the compute node that received all the data from the other compute nodes is a receiving process. The compute node on which such an originating or receiving process runs is referred to as a logical root.

Most collective operations are variations or combinations of four basic operations: broadcast, gather, scatter, and reduce. The interfaces for these collective operations are defined in the MPI standards promulgated by the MPI Forum. Algorithms for executing collective operations, however, are not defined in the MPI standards. In a broadcast operation, all processes specify the same root process, whose buffer contents will be sent. Processes other than the root specify receive buffers. After the operation, all buffers contain the message from the root process.

A scatter operation, like the broadcast operation, is also a one-to-many collective operation. In a scatter operation, the logical root divides data on the root into segments and distributes a different segment to each compute node in the operational group (**132**). In scatter operation, all processes typically specify the same receive count. The send arguments are only significant to the root process, whose buffer actually contains sendcount*N elements of a given datatype, where N is the number of processes in the given group of compute nodes. The send buffer is divided and dispersed to all processes (including the process on the logical root). Each compute node is assigned a sequential identifier termed a 'rank.' After the operation, the root has sent sendcount data elements to each process in increasing rank order. Rank 0 receives the first sendcount data elements from the send buffer. Rank 1 receives the second sendcount data elements from the send buffer, and so on.

A gather operation is a many-to-one collective operation that is a complete reverse of the description of the scatter operation. That is, a gather is a many-to-one collective operation in which elements of a datatype are gathered from the ranked compute nodes into a receive buffer in a root node.

A reduction operation is also a many-to-one collective operation that includes an arithmetic or logical function performed on two data elements. All processes specify the same 'count' and the same arithmetic or logical function. After the reduction, all processes have sent count data elements from compute node send buffers to the root process. In a reduction operation, data elements from corresponding send buffer locations are combined pair-wise by arithmetic or logical operations to yield a single corresponding element in the root process' receive buffer. Application specific reduction operations can be defined at runtime. Parallel communications libraries may support predefined operations. MPI, for example, provides the following predefined reduction operations:

MPI_MAX	maximum
MPI_MIN	minimum
MPI_SUM	sum
MPI_PROD	product
MPI_LAND	logical and
MPI_BAND	bitwise and
MPI_LOR	logical or
MPI_BOR	bitwise or
MPI_LXOR	logical exclusive or
MPI_BXOR	bitwise exclusive or

In addition to compute nodes, the parallel computer (**100**) includes input/output ('I/O') nodes (**110**, **114**) coupled to compute nodes (**102**) through the global combining network (**106**). The compute nodes (**102**) in the parallel computer (**100**) may be partitioned into processing sets such that each compute node in a processing set is connected for data communications to the same I/O node. Each processing set, therefore, is composed of one I/O node and a subset of compute nodes (**102**). The ratio between the number of compute nodes to the number of I/O nodes in the entire system typically depends on the hardware configuration for the parallel computer (**102**). For example, in some configurations, each processing set may be composed of eight compute nodes and one I/O node. In some other configurations, each processing set may be composed of sixty-four compute nodes and one I/O node. Such example are for explanation only, however, and not for limitation. Each I/O node provides I/O services between compute nodes (**102**) of its processing set and a set of I/O devices. In the example of FIG. **1**, the I/O nodes (**110**, **114**) are connected for data communications I/O devices (**118**, **120**, **122**) through local area network ('LAN') (**130**) implemented using high-speed Ethernet.

The parallel computer (**100**) of FIG. **1** also includes a service node (**116**) coupled to the compute nodes through one of the networks (**104**). Service node (**116**) provides services common to pluralities of compute nodes, administering the configuration of compute nodes, loading programs into the compute nodes, starting program execution on the compute nodes, retrieving results of program operations on the compute nodes, and so on. Service node (**116**) runs a service application (**124**) and communicates with users (**128**) through a service application interface (**126**) that runs on computer terminal (**122**).

The parallel computer (**100**) of FIG. **1** operates generally for parallel application checkpoint image compression in accordance with embodiments of the present invention. In the example of FIG. **1** each of the compute nodes (**102**) of the operational group (**132**) may execute a separate instance of a parallel application, or a task, as described below. As such, the parallel computer (**100**) of FIG. **1** operates for parallel application checkpoint image compression in accordance with embodiments of the present invention by generating, by each task of the parallel application, an image for checkpointing the parallel application. An image management task, which may be implemented by any of the tasks of the operational group or may be a separate task not included in the operational group, may then select one of the images as a base template image and construct a binary radix tree. Constructing the binary radix tree may include storing differences between each task's image and the base template image in the binary radix tree. The image management task may then store as a checkpoint for the parallel application, the binary radix tree and the base template image, without storing every task's image.

A tree data structure is typically composed of a plurality of nodes logically connected in a manner that resembles an inverted tree. In many tree data structures, the key values or entries in the tree are stored in the various nodes of the tree. Leaf nodes are nodes in the tree that have no children. By contrast, the root node of the tree is a node in the tree that has no parent. Nodes logically positioned between the root node and the leaf nodes are referred to as limb nodes and have both a parent and a child.

A radix tree (also patricia trie or radix trie or compact prefix tree) is a space-optimized trie data structure where each node with only one child is merged with its child. A BinaryRadixTree is a memory-optimized radix tree that is intended to efficiently store a mapping from binary values to long values

Rather than storing one image for each task where there may be significant duplication of data between images—because each task is an identical separate instance executing at roughly the same speed on similar hardware with similar data inputs—the image management task utilizes the binary radix tree to maintain only one image (the base template) along with differences (represented in the tree) between other images and the base template image.

Parallel application checkpoint image compression in a parallel computer according to embodiments of the present invention is generally implemented on a parallel computer that includes a plurality of compute nodes organized for collective operations through at least one data communications network. In fact, such computers may include thousands of such compute nodes. Each compute node is in turn itself a kind of computer composed of one or more computer processing cores, its own computer memory, and its own input/output adapters. For further explanation, therefore, FIG. **2** sets forth a block diagram of an example compute node (**102**) useful in a parallel computer capable of parallel application checkpoint image compression according to embodiments of the present invention. The compute node (**102**) of FIG. **2** includes a plurality of processing cores (**165**) as well as RAM (**156**). The processing cores (**165**) of FIG. **2** may be configured on one or more integrated circuit dies. Processing cores (**165**) are connected to RAM (**156**) through a high-speed memory bus (**155**) and through a bus adapter (**194**) and an extension bus (**168**) to other components of the compute node. Stored in RAM (**156**) is a parallel application program (**159**), a module of computer program instructions that carries out parallel, user-level data processing using parallel algorithms.

In the example of FIG. **2**, the parallel application (**226**) may include a number of parallel tasks (**228**). In some embodiments, the tasks may be implemented as ranks for MPI-like configurations. Further, the parallel application (**226**) and tasks (**228**) may be distributed amongst many nodes in the parallel computer. The tasks (**228**) may be organized into an operational group in order to carry out, or support, collective operations and communications amongst the tasks of the operational group.

The parallel application (**226**) and the tasks (**228**), in the example of FIG. **2** may also be configured for checkpoint image compression in the parallel computer. Each tasks (**228**) may generate an image (**230**) for checkpointing the parallel application. An image management task, which may be implemented as any of the parallel application tasks (**228**) or another task that is not shown in FIG. **2**, may select one of the images (**230**) as a base template image (**232**). The image management task may then construct a binary radix tree (**234**) by storing differences between each task's (**228**) image (**230**) and the base template image (**232**) in the binary radix tree (**234**). The image management task may then store, as a checkpoint for the parallel application, the binary radix tree (**234**) and the base template image (**232**), without storing every task's image.

Also stored RAM (**156**) is a parallel communications library (**161**), a library of computer program instructions that carry out parallel communications among compute nodes, including point-to-point operations as well as collective operations. A library of parallel communications routines may be developed from scratch for use in systems according to embodiments of the present invention, using a traditional programming language such as the C programming language, and using traditional programming methods to write parallel communications routines that send and receive data among nodes on two independent data communications networks. Alternatively, existing prior art libraries may be improved to operate according to embodiments of the present invention. Examples of prior-art parallel communications libraries include the 'Message Passing Interface' ('MPI') library and the 'Parallel Virtual Machine' ('PVM') library.

Also stored in RAM (**156**) is an operating system (**162**), a module of computer program instructions and routines for an application program's access to other resources of the compute node. It is typical for an application program and parallel communications library in a compute node of a parallel computer to run a single thread of execution with no user login and no security issues because the thread is entitled to complete access to all resources of the node. The quantity and complexity of tasks to be performed by an operating system on a compute node in a parallel computer therefore are smaller and less complex than those of an operating system on a serial computer with many threads running simultaneously. In addition, there is no video I/O on the compute node (**102**) of FIG. **2**, another factor that decreases the demands on the operating system. The operating system (**162**) may therefore be quite lightweight by comparison with operating systems of general purpose computers, a pared down version as it were, or an operating system developed specifically for operations on a particular parallel computer. Operating systems that may usefully be improved, simplified, for use in a compute node include UNIX™, Linux™, Windows XP™, AIX™, IBM's i5/OS™, and others as will occur to those of skill in the art.

The example compute node (**102**) of FIG. **2** includes several communications adapters (**172**, **176**, **180**, **188**) for implementing data communications with other nodes of a parallel computer. Such data communications may be carried out serially through RS-232 connections, through external buses such as USB, through data communications networks such as IP networks, and in other ways as will occur to those of skill in the art. Communications adapters implement the hardware level of data communications through which one computer sends data communications to another computer, directly or through a network. Examples of communications adapters useful in apparatus for parallel application checkpoint image compression include modems for wired communications, Ethernet (IEEE 802.3) adapters for wired network communications, and 802.11b adapters for wireless network communications.

The data communications adapters in the example of FIG. **2** include a Gigabit Ethernet adapter (**172**) that couples example compute node (**102**) for data communications to a Gigabit Ethernet (**174**). Gigabit Ethernet is a network transmission standard, defined in the IEEE 802.3 standard, that provides a data rate of 1 billion bits per second (one gigabit). Gigabit Ethernet is a variant of Ethernet that operates over multimode fiber optic cable, single mode fiber optic cable, or unshielded twisted pair.

The data communications adapters in the example of FIG. **2** include a JTAG Slave circuit (**176**) that couples example compute node (**102**) for data communications to a JTAG Master circuit (**178**). JTAG is the usual name used for the IEEE 1149.1 standard entitled Standard Test Access Port and Boundary-Scan Architecture for test access ports used for testing printed circuit boards using boundary scan. JTAG is so widely adapted that, at this time, boundary scan is more or less synonymous with JTAG. JTAG is used not only for printed circuit boards, but also for conducting boundary scans of integrated circuits, and is also useful as a mechanism for debugging embedded systems, providing a convenient alternative access point into the system. The example compute node of FIG. **2** may be all three of these: It typically includes one or more integrated circuits installed on a printed circuit board and may be implemented as an embedded system having its own processing core, its own memory, and its own I/O capability. JTAG boundary scans through JTAG Slave (**176**) may efficiently configure processing core registers and memory in compute node (**102**) for use in dynamically reassigning a connected node to a block of compute nodes useful in systems for parallel application checkpoint image compression to embodiments of the present invention.

The data communications adapters in the example of FIG. **2** include a Point-To-Point Network Adapter (**180**) that couples example compute node (**102**) for data communications to a network (**108**) that is optimal for point-to-point message passing operations such as, for example, a network configured as a three-dimensional torus or mesh. The Point-To-Point Adapter (**180**) provides data communications in six directions on three communications axes, x, y, and z, through six bidirectional links: +x (**181**), −x (**182**), +y (**183**), −y (**184**), +z (**185**), and −z (**186**).

The data communications adapters in the example of FIG. **2** include a Global Combining Network Adapter (**188**) that couples example compute node (**102**) for data communications to a global combining network (**106**) that is optimal for collective message passing operations such as, for example, a network configured as a binary tree. The Global Combining Network Adapter (**188**) provides data communications through three bidirectional links for each global combining network (**106**) that the Global Combining Network Adapter (**188**) supports. In the example of FIG. **2**, the Global Combining Network Adapter (**188**) provides data communications through three bidirectional links for global combining network (**106**): two to children nodes (**190**) and one to a parent node (**192**).

The example compute node (**102**) includes multiple arithmetic logic units ('ALUs'). Each processing core (**165**) includes an ALU (**166**), and a separate ALU (**170**) is dedicated to the exclusive use of the Global Combining Network Adapter (**188**) for use in performing the arithmetic and logical functions of reduction operations, including an allreduce operation. Computer program instructions of a reduction routine in a parallel communications library (**161**) may latch an instruction for an arithmetic or logical function into an instruction register (**169**). When the arithmetic or logical function of a reduction operation is a 'sum' or a 'logical OR,' for example, the collective operations adapter (**188**) may execute the arithmetic or logical operation by use of the ALU (**166**) in the processing core (**165**) or, typically much faster, by use of the dedicated ALU (**170**) using data provided by the nodes (**190**, **192**) on the global combining network (**106**) and data provided by processing cores (**165**) on the compute node (**102**).

Often when performing arithmetic operations in the global combining network adapter (**188**), however, the global combining network adapter (**188**) only serves to combine data received from the children nodes (**190**) and pass the result up the network (**106**) to the parent node (**192**). Similarly, the global combining network adapter (**188**) may only serve to transmit data received from the parent node (**192**) and pass the data down the network (**106**) to the children nodes (**190**). That is, none of the processing cores (**165**) on the compute node (**102**) contribute data that alters the output of ALU (**170**), which is then passed up or down the global combining network (**106**). Because the ALU (**170**) typically does not output any data onto the network (**106**) until the ALU (**170**) receives input from one of the processing cores (**165**), a processing core (**165**) may inject the identity element into the dedicated ALU (**170**) for the particular arithmetic operation being perform in the ALU (**170**) in order to prevent alteration of the output of the ALU (**170**). Injecting the identity element into the ALU, however, often consumes numerous processing cycles. To further enhance performance in such cases, the example compute node (**102**) includes dedicated hardware (**171**) for injecting identity elements into the ALU (**170**) to reduce the amount of processing core resources required to prevent alteration of the ALU output. The dedicated hardware (**171**) injects an identity element that corresponds to the particular arithmetic operation performed by the ALU. For example, when the global combining network adapter (**188**) performs a bitwise OR on the data received from the children nodes (**190**), dedicated hardware (**171**) may inject zeros into the ALU (**170**) to improve performance throughout the global combining network (**106**).

For further explanation, FIG. **3** sets forth a block diagram of an example Point-To-Point Adapter (**180**) useful in systems for parallel application checkpoint image compression according to embodiments of the present invention. The Point-To-Point Adapter (**180**) is designed for use in a data communications network optimized for point-to-point operations, a network that organizes compute nodes in a three-dimensional torus or mesh. The Point-To-Point Adapter (**180**) in the example of FIG. **3** provides data communication along an x-axis through four unidirectional data communications links, to and from the next node in the −x direction (**182**) and to and from the next node in the +x direction (**181**). The Point-To-Point Adapter (**180**) of FIG. **3** also provides data communication along a y-axis through four unidirectional data communications links, to and from the next node in the −y direction (**184**) and to and from the next node in the +y direction (**183**). The Point-To-Point Adapter (**180**) of FIG. **3** also provides data communication along a z-axis through four unidirectional data communications links, to and from the next node in the −z direction (**186**) and to and from the next node in the +z direction (**185**).

For further explanation, FIG. **4** sets forth a block diagram of an example Global Combining Network Adapter (**188**) useful in systems for parallel application checkpoint image compression according to embodiments of the present invention.

The Global Combining Network Adapter (**188**) is designed for use in a network optimized for collective operations, a network that organizes compute nodes of a parallel computer in a binary tree. The Global Combining Network Adapter (**188**) in the example of FIG. **4** provides data communication to and from children nodes of a global combining network through four unidirectional data communications links (**190**), and also provides data communication to and from a parent node of the global combining network through two unidirectional data communications links (**192**).

For further explanation, FIG. **5** sets forth a line drawing illustrating an example data communications network (**108**) optimized for point-to-point operations useful in systems capable of parallel application checkpoint image compression according to embodiments of the present invention. In the example of FIG. **5**, dots represent compute nodes (**102**) of a parallel computer, and the dotted lines between the dots represent data communications links (**103**) between compute nodes. The data communications links are implemented with point-to-point data communications adapters similar to the one illustrated for example in FIG. **3**, with data communications links on three axis, x, y, and z, and to and fro in six directions +x (**181**), −x (**182**), +y (**183**), −y (**184**), +z (**185**), and −z (**186**). The links and compute nodes are organized by this data communications network optimized for point-to-point operations into a three dimensional mesh (**105**). The mesh (**105**) has wrap-around links on each axis that connect the outermost compute nodes in the mesh (**105**) on opposite sides of the mesh (**105**). These wrap-around links form a torus (**107**). Each compute node in the torus has a location in the torus that is uniquely specified by a set of x, y, z coordinates. Readers will note that the wrap-around links in the y and z directions have been omitted for clarity, but are configured in a similar manner to the wrap-around link illustrated in the x direction. For clarity of explanation, the data communications network of FIG. **5** is illustrated with only 27 compute nodes, but readers will recognize that a data communications network optimized for point-to-point operations for use in parallel application checkpoint image compression in accordance with embodiments of the present invention may contain only a few compute nodes or may contain thousands of compute nodes. For ease of explanation, the data communications network of FIG. **5** is illustrated with only three dimensions, but readers will recognize that a data communications network optimized for point-to-point operations for use in parallel application checkpoint image compression in accordance with embodiments of the present invention may in facet be implemented in two dimensions, four dimensions, five dimensions, and so on. Several supercomputers now use five dimensional mesh or torus networks, including, for example, IBM's Blue Gene Q™.

For further explanation, FIG. **6** sets forth a line drawing illustrating an example global combining network (**106**) useful in systems capable of parallel application checkpoint image compression according to embodiments of the present invention. The example data communications network of FIG. **6** includes data communications links (**103**) connected to the compute nodes so as to organize the compute nodes as a tree. In the example of FIG. **6**, dots represent compute nodes (**102**) of a parallel computer, and the dotted lines (**103**) between the dots represent data communications links between compute nodes. The data communications links are implemented with global combining network adapters similar to the one illustrated for example in FIG. **4**, with each node typically providing data communications to and from two children nodes and data communications to and from a parent node, with some exceptions. Nodes in the global combining network (**106**) may be characterized as a physical root node (**202**), branch nodes (**204**), and leaf nodes (**206**). The physical root (**202**) has two children but no parent and is so called because the physical root node (**202**) is the node physically configured at the top of the binary tree. The leaf nodes (**206**) each has a parent, but leaf nodes have no children. The branch nodes (**204**) each has both a parent and two children. The links and compute nodes are thereby organized by this data communications network optimized for collective operations into a binary tree (**106**). For clarity of explanation, the data communications network of FIG. **6** is illustrated with only 31 compute nodes, but readers will recognize that a global combining network (**106**) optimized for collective operations for use in parallel application checkpoint image compression in accordance with embodiments of the present invention may contain only a few compute nodes or may contain thousands of compute nodes.

In the example of FIG. **6**, each node in the tree is assigned a unit identifier referred to as a 'rank' (**250**). The rank actually identifies a task or process that is executing a parallel operation according to embodiments of the present invention. Using the rank to identify a node assumes that only one such task is executing on each node. To the extent that more than one participating task executes on a single node, the rank identifies the task as such rather than the node. A rank uniquely identifies a task's location in the tree network for use in both point-to-point and collective operations in the tree network. The ranks in this example are assigned as integers beginning with 0 assigned to the root tasks or root node (**202**), 1 assigned to the first node in the second layer of the tree, 2 assigned to the second node in the second layer of the tree, 3 assigned to the first node in the third layer of the tree, 4 assigned to the second node in the third layer of the tree, and so on. For ease of illustration, only the ranks of the first three layers of the tree are shown here, but all compute nodes in the tree network are assigned a unique rank.

For further explanation, FIG. **7** sets forth a line drawing illustrating an example binary radix tree useful in parallel application checkpoint image compression according to embodiments of the present invention. In the example binary radix tree (**234**) of FIG. **7**, three separate images are represented. Each of these images has been generated from a separate task. In all then, four tasks have created checkpoints: one task providing the base template image and three tasks providing images having some difference from the base template image.

In the example of FIG. **7**, three has an index (**702**) that operates as a root for all images represented by the tree. The parent nodes (**738**) of the tree set forth image identifiers of the three images: referred to here as N (**704**), N+1 (**706**), and N+2 (**708**).

The leaf nodes (**740**) represent, for each image, a memory offset at which a difference between the image and the base template image begins as well as the actual value of the difference (referred to in FIG. **7** as 'difference data'). Image ID=N (**704**) has a difference at offset **100** (**710**), offset **110** (**712**), and offset **210** (**714**). Each of the offsets corresponds to difference data (**724**, **726**, **728**). The difference data may be of any length in some embodiments. That is, each difference data may be of the same or different length.

Image ID=N+1 (**706**) has a difference at offset **100** (**716**) with difference data (**730**) and a difference at offset **250** (**718**) with difference data (**732**). Image ID=N+2 (**708**) has a difference at offset **280** (**720**) with difference data (**734**) and a difference at offset **340** (**722**) with difference data (**736**).

To reconstruct an image for a task having the identifier N upon a restart of the parallel application from the checkpoint represented by the base image template and the tree, one or more tasks may query a tree management process for all values of image identifier N. The tree management process may return a set of values that includes the image identifier (**704**) and each pair of offset (**710**,**712**, **714**) and the corresponding difference data (**724**, **726**, **728**). A similar query may be made for each image identifier. Once these values are received, the image management process may copy the base template image and modify, beginning at each offset retrieved from the tree, the copied image by replacing that portion of the image with the difference data.

For further explanation, FIG. **8** sets forth a flow chart illustrating an example method for parallel application checkpoint image compression according to embodiments of the present invention. In the method of FIG. **8**, the parallel computer includes a plurality of compute nodes where each compute node may be configured to execute one or more parallel tasks of the parallel application. The tasks may be organized into an operational group. In embodiments in which the parallel tasks are implemented as a MPI-like ranks, the operational group may be an MPI communicator.

The method of FIG. **8** includes generating (**802**), by each task of the parallel application, an image for checkpointing the parallel application. Each task may generate an image in a variety of ways. A task for example may store all current application state such as global variables, program counter, application call stack, and so on.

The method of FIG. **8** also includes selecting (**804**), by an image management task, one of the images as a base template image. An image management task may be one of the tasks of the operational group or another task not included in the operational group. Such a task may select one of the tasks' images as the base template image in a variety of ways in accordance with predefined selection criteria. Such predefined selection criteria may be configured by a user and may specify, for example, selection of the smallest image as the base template image, the largest template as the base template image, the image that, if selected as the base template image, results in the fewest number of differences from other images, and so on as will occur to readers of skill in the art.

The method of FIG. **8** also includes constructing (**806**), by the image management task, a binary radix tree. In the example of FIG. **8**, constructing (**806**) a binary radix tree is carried out by storing (**810**) differences between each task's image and the base template image in the binary radix tree. That is, the image management task may create a tree like data structure that includes one or more parent nodes. Each parent node is utilized to store an image identifier. The tree data structure may also include one or more leaf nodes, with each leaf node being a child of a parent node and including an offset representing a beginning address at which an image includes a difference between the image and the base template image and difference data representing the difference between the image and the base template image.

The method of FIG. **8** also includes storing (**808**), by the image management task as a checkpoint for the parallel application, the binary radix tree and the base template image, without storing every task's image. Storing (**808**) the binary radix tree and the base template image may be carried out in variety of ways including, storing the data structure (the tree) and the base template image in secondary storage such as a disk drive, a storage area network, or the like. Such a storage location may be identified by a management application. It is noted that such a management application may also be responsible for initiating the checkpoint. Such a management application initiate the checkpointing at the behest of a user or periodically and without any user interaction or in some combination.

For further explanation, FIG. **9** sets forth a flow chart illustrating another example method for parallel application checkpoint image compression according to embodiments of the present invention. The method of FIG. **9** is similar to the method of FIG. **8** in that the method of FIG. **9** also includes generating (**802**) an image by each task; selecting (**804**) one of the images as a base template image; constructing (**806**) a binary radix tree; and storing (**808**) the binary radix tree and the base template image, without storing every task's image.

The method of FIG. **9** differs from the method of FIG. **8**, however, in that, upon a restart of the parallel application from the checkpoint, the method of FIG. **9** includes reconstructing (**902**) each task's image from the base template and the differences, stored in the binary radix tree, between the task's image and the base template stored in the binary radix tree. In the method of FIG. **9**, reconstructing (**902**) each task's image may be carried out by processing (**904**) the binary radix tree in parallel by a plurality of tasks. That is, a first set of one or more of tasks may be configured to reconstruct an image for a first task, while a second set of one or more tasks may be configured to reconstruct an image for a second task, and so for each image to be reconstructed. This is but one way, among many, in which parallelism may be used to both reconstruct and create the checkpoint for the parallel application.

For further explanation, FIG. **10** sets forth a flow chart illustrating another example method for parallel application checkpoint image compression according to embodiments of the present invention. The method of FIG. **10** is similar to the method of FIG. **8** in that the method of FIG. **10** also includes generating (**802**) an image by each task; selecting (**804**) one of the images as a base template image; constructing (**806**) a binary radix tree; and storing (**808**) the binary radix tree and the base template image, without storing every task's image.

The method of FIG. **10** differs from the method of FIG. **8**, however, in the method of FIG. **10** identifying (**1002**), by each task in parallel with the other tasks, one or more differences between the task's image and the base template image and providing (**1004**), by each task to the image management task, the identified differences. Identifying (**1002**), by each task in parallel with the other tasks, one or more differences between the task's image and the base template image may be carried out in a variety of ways including for example, executing one or more bitwise XOR comparison between the binary version of the base template image and the task's image to identify any differences. Then, the image management task may operate as a root node in a collective operation in order to gather the offsets and differences from the tasks in the operational groups. Such a collective operation may include, for example, an MPI gather operation.

As will be appreciated by one skilled in the art, aspects of the present invention may be embodied as a system, method or computer program product. Accordingly, aspects of the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "circuit," "module" or "system." Furthermore, aspects of the present invention may take the form of a computer program product embodied in one or more computer readable medium(s) having computer readable program code embodied thereon.

Any combination of one or more computer readable medium(s) may be utilized. The computer readable medium may be a computer readable transmission medium or a computer readable storage medium. A computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer readable storage medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device.

A computer readable transmission medium may include a propagated data signal with computer readable program code embodied therein, for example, in baseband or as part of a carrier wave. Such a propagated signal may take any of a variety of forms, including, but not limited to, electro-magnetic, optical, or any suitable combination thereof. A computer readable transmission medium may be any computer readable medium that is not a computer readable storage medium and that can communicate, propagate, or transport a program for use by or in connection with an instruction execution system, apparatus, or device.

Program code embodied on a computer readable medium may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fiber cable, RF, etc., or any suitable combination of the foregoing.

Computer program code for carrying out operations for aspects of the present invention may be written in any combination of one or more programming languages, including an object oriented programming language such as Java, Smalltalk, C++ or the like and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

Aspects of the present invention are described above with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer readable medium that can direct a computer, other programmable data processing apparatus, or other devices to function in a particular manner, such that the instructions stored in the computer readable medium produce an article of manufacture including instructions which implement the function/act specified in the flowchart and/or block diagram block or blocks.

The computer program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatus or other devices to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

It will be understood from the foregoing description that modifications and changes may be made in various embodiments of the present invention without departing from its true spirit. The descriptions in this specification are for purposes of illustration only and are not to be construed in a limiting sense. The scope of the present invention is limited only by the language of the following claims.

What is claimed is:

1. A method of parallel application checkpoint image compression in a parallel computer, the parallel computer comprising a plurality of compute nodes, each compute node configured to execute one or more parallel tasks of the parallel application, the parallel tasks organized into an operational group, the method comprising:

generating, by each task of the parallel application, an image for checkpointing the parallel application;

selecting, by an image management task, one of the images as a base template image;

constructing, by the image management task, a binary radix tree, including storing differences between each task's image and the base template image in the binary radix tree; and

storing, by the image management task as a checkpoint for the parallel application, the binary radix tree and the base template image, without storing every task's image.

2. The method of claim **1**, further comprising:

upon a restart of the parallel application from the checkpoint, reconstructing each task's image from the base template and the differences, stored in the binary radix tree, between the task's image and the base template stored in the binary radix tree.

3. The method of claim **2**, wherein:

reconstructing each task's image from the base template and the differences, stored in the binary radix tree, between the task's image and the base template stored in the binary radix tree further comprises processing the binary radix tree in parallel by a plurality of tasks.

4. The method of claim **1**, wherein the binary radix tree further comprises:

one or more parent nodes, each parent node comprising an image identifier; and

one or more leaf nodes, each leaf node being a child of a parent node and including an offset representing a beginning address at which an image includes a difference between the image and the base template image and difference data representing the difference between the image and the base template image.

5. The method of claim **1**, wherein storing differences between each task's image and the base template image in the binary radix tree further comprises:

identifying, by each task in parallel with the other tasks, one or more differences between the task's image and the base template image; and

providing, by each task to the image management task, the identified differences.

6. The method of claim **1**, wherein constructing, by the image management task, a binary radix tree further comprises:

constructing the binary radix tree in parallel by a plurality of tasks and the image management task.

7. An apparatus for parallel application checkpoint image compression in a parallel computer, the parallel computer comprising a plurality of compute nodes, each compute node configured to execute one or more parallel tasks of the parallel application, the parallel tasks organized into an operational group, the apparatus comprising a computer processor, a computer memory operatively coupled to the computer processor, the computer memory having disposed within it computer program instructions that, when executed by the computer processor, cause the apparatus to carry out the steps of:

generating, by each task of the parallel application, an image for checkpointing the parallel application;

selecting, by an image management task, one of the images as a base template image;

constructing, by the image management task, a binary radix tree, including storing differences between each task's image and the base template image in the binary radix tree; and

storing, by the image management task as a checkpoint for the parallel application, the binary radix tree and the base template image, without storing every task's image.

8. The apparatus of claim **7**, further comprising computer program instructions that, when executed, cause the apparatus to carry out the step of:

upon a restart of the parallel application from the checkpoint, reconstructing each task's image from the base template and the differences, stored in the binary radix tree, between the task's image and the base template stored in the binary radix tree.

9. The apparatus of claim **8**, wherein:

reconstructing each task's image from the base template and the differences, stored in the binary radix tree, between the task's image and the base template stored in the binary radix tree further comprises processing the binary radix tree in parallel by a plurality of tasks.

10. The apparatus of claim **7**, wherein the binary radix tree further comprises:

one or more parent nodes, each parent node comprising an image identifier; and

one or more leaf nodes, each leaf node being a child of a parent node and including an offset representing a beginning address at which an image includes a difference between the image and the base template image and difference data representing the difference between the image and the base template image.

11. The apparatus of claim **7**, wherein storing differences between each task's image and the base template image in the binary radix tree further comprises:

identifying, by each task in parallel with the other tasks, one or more differences between the task's image and the base template image; and

providing, by each task to the image management task, the identified differences.

12. The apparatus of claim **7**, wherein constructing, by the image management task, a binary radix tree further comprises:

constructing the binary radix tree in parallel by a plurality of tasks and the image management task.

13. A computer program product for parallel application checkpoint image compression in a parallel computer, the parallel computer comprising a plurality of compute nodes, each compute node configured to execute one or more parallel tasks of the parallel application, the parallel tasks organized into an operational group, the computer program product disposed upon a computer readable non-transitory medium, the computer program product comprising computer program instructions that, when executed, cause a parallel computer to carry out the steps of:

generating, by each task of the parallel application, an image for checkpointing the parallel application;

selecting, by an image management task, one of the images as a base template image;

constructing, by the image management task, a binary radix tree, including storing differences between each task's image and the base template image in the binary radix tree; and

storing, by the image management task as a checkpoint for the parallel application, the binary radix tree and the base template image, without storing every task's image.

14. The computer program product of claim **13**, further comprising computer program instructions that, when executed, cause the parallel computer to carry out the step of:

upon a restart of the parallel application from the checkpoint, reconstructing each task's image from the base template and the differences, stored in the binary radix tree, between the task's image and the base template stored in the binary radix tree.

15. The computer program product of claim **14**, wherein:

reconstructing each task's image from the base template and the differences, stored in the binary radix tree, between the task's image and the base template stored in the binary radix tree further comprises processing the binary radix tree in parallel by a plurality of tasks.

16. The computer program product of claim **13**, wherein the binary radix tree further comprises:

one or more parent nodes, each parent node comprising an image identifier; and

one or more leaf nodes, each leaf node being a child of a parent node and including an offset representing a beginning address at which an image includes a difference between the image and the base template image and difference data representing the difference between the image and the base template image.

17. The computer program product of claim **13**, wherein storing differences between each task's image and the base template image in the binary radix tree further comprises:

identifying, by each task in parallel with the other tasks, one or more differences between the task's image and the base template image; and

providing, by each task to the image management task, the identified differences.

18. The computer program product of claim **13**, wherein constructing, by the image management task, a binary radix tree further comprises:

constructing the binary radix tree in parallel by a plurality of tasks and the image management task.

* * * * *